CS575 Parallel Computing Project#3
False Sharing

Shangjia Dong

[1]School of Civil and Construction Engineering, Oregon State University

dongs@oregonstate.edu

1.

Question: Tell what machine you ran this on.

I ran this on rabbit.

- 2 E5-2630 Xeon Processors
- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- CPU(s): 32
- Thread(s) per core: 2; Core(s) per socket: 8
- Socket(s): 2
- 64 GB of memory; 2 TB of disk

2.

Question: Create a table with your results.

Table 1. Fix 1 Performance (megaSum/Sec) VS. NUMPAD

	0	1	2	3	4	5	6	7
#Thread = 1	185.99	184.92	183.25	182.39	183.79	181.69	177.36	182.49
#Thread = 2	97.64	97.28	99.5	106.27	101.6	161.66	152.67	364.39
#Thread = 4	92.85	92.02	90.54	104.12	93.94	128.07	113.25	190.11
	8	9	10	11	12	13	14	15
#Thread = 1	183.8	183.79	171.39	183.25	168.98	177.39	181.19	182.41
#Thread = 2	367.16	367.28	348.3	365.92	337.39	354.31	361.86	364.49
#Thread = 4	173.29	187.89	204.41	199.59	197.52	194.78	190.34	716.99

Table 2. Fix #2 Performance (megaSum/Sec)

	Performance
#Thread = 1	185.02
#Thread = 2	369.04
#Thread = 4	719.53

3.

Question: Draw a graph. The X axis will be NUMPAD, i.e., the amount of integers used to pad the structure. The Y axis will be the performance in whatever units you sensibly choose. There should be at least 6 curves shown together on those axes:

- 1-3: Using padding with 1, 2, and 4 threads. (Fix 1)
- 4-6: Using a private variable with 1, 2, and 4 threads. (Fix 2)

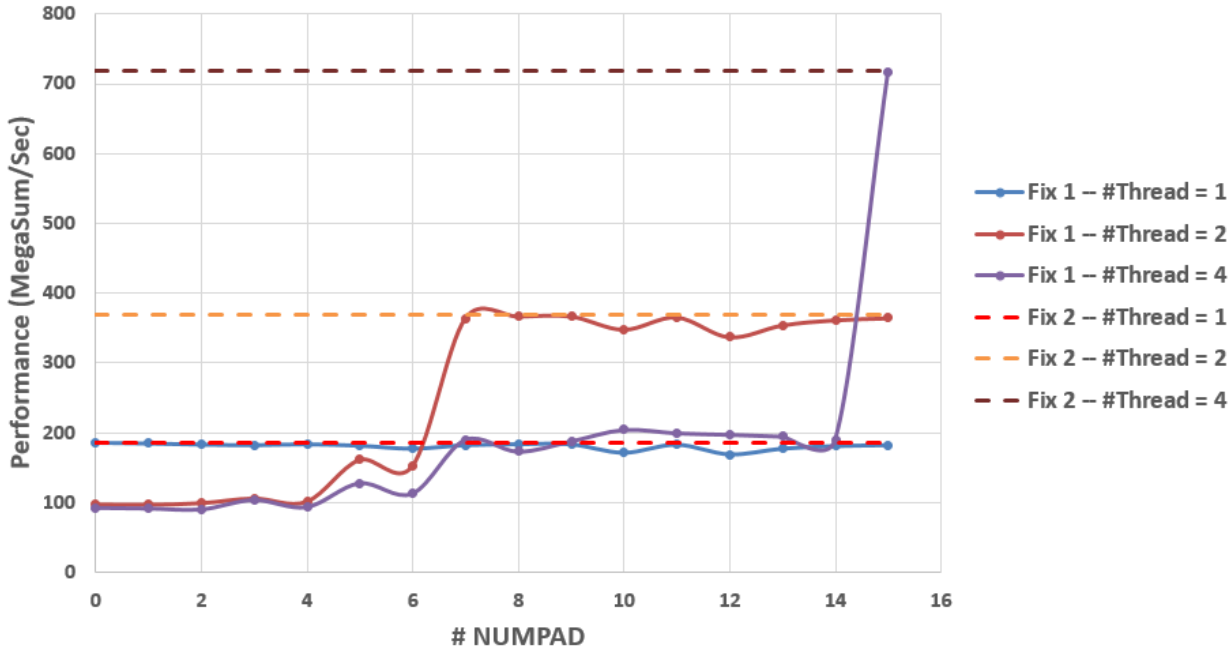

Figure 1. Fix1 and Fix 2 performance summary

4.

Question: What patterns are you seeing in the performance?

Result: When there is only thread, the performance stays fairly steady no matter how the NUMPAD changes.

When there is two threads, the performance is lower than the case when there is only one thread at the beginning of padding, but the performance increases at NUMPAD = 5 and 7. And the performance stays the same after 7.

When there is four threads, the performance is lower than the case when there is only one thread at the beginning of padding, but the performance increases at NUMPAD = 5, 7, and 14. After NUMPAD = 7, the performance is better than the performance when there is only one thread.

5.

Question: Why do you think it is behaving this way?

Result: For the Fix 1: When there is only one thread, there is no false sharing. When add more threads in, there is more cores than cache lines, then multiple threads are

gonna read and write on the same cache line, the false sharing starts to happen: one thread is writing while another one is trying to read, when they notice the variable is invalid, they have to flush back to memory and read it again, which leads to a performance drop. When the pad reaches to a point, the performance starts to getting better. That's because successive Array elements are forced onto different cache lines, so less (or no) cache line conflicts exist. The best case is when each thread reads from one cache line, then the performance is huge, like when NUMPAD = 15 and Threads = 4.

Look at the figure, since cache line has 64 bytes, and the Array elements is defined as float type, so it is 4 bytes. Then there is totally 16 spots that can be used to store an Array element. When the thread = 2, the best performance occurs when NUMPAD = 7, although there are two Array elements on cache line, but each threads got they own cache line to read from. Similarly, When the thread = 4, the best performance occurs when the four Array elements evenly spread to four cache lines (NUMPAD = 15).

For Fix 2: When we use local variables, instead of contiguous array locations, that will spread our writes out in memory, and to different cache lines. When a localized temporary variable is created in each core's stack area, little or no cache line conflict exists. That's why when we increase the threads number, the performance increases.